01背包dp

2019年2月27日　　17:45

-
- 01背包

一. 基本思想
   1. 数据：物品数n、体积v、价值w、背包容量m
   2. 变量
      i. 阶段i：处理的第i个物品
      ii. 状态j：当前体积
      iii. 存储的数据：价值w
   3. 初值：w[0][0]为0，其余负无穷（memset(w,0x80,sizeof(w));w[0][0]=0;）（可根据最终结果是不是负数来判断有没有恰能凑出这个体积的货物，或在转移时判断=右边是负值就不转）
   4. 转移方程
      i. w[i][j]=max(w[i-1][j],　　//不选择第i个物品
         w[i-1][j-v[i]]+w[i]);　　//选择第i个物品
      ii. 循环i：[1,n]（物品编号从1开始时，而且这样不用处理越界）
      iii. 循环j：[v[i],m]（否则会越界）

二. 滚动数组法
   1. 思想：每一阶段i只与前一阶段i-1有关，只需要存储两行，可利用奇偶
   2. 实现方法：在阶段i和i-1改为i&1和(i-1)&1
   3. for__(i,1,n){

```
    for__(j,0,m)      //别忘了给j<vi的状态也转移状态
        w[i&1][j]=w[(i-1)&1][j];
    for__(j,v[i],m)
        w[i&1][j]=max(w[i&1][j],          //不选择第i个物品
            w[(i-1)&1][j-v[i]]+w[i]);  //选择第i个物品
}
```

三. 一维逆序法
   1. 思想：每一阶段i只与前一阶段i-1有关，可以直接覆盖在唯一的一行
   2. 实现方法：因为转移方程是从j-vi转移到j，所以j逆序循环覆盖的话就可以做到每次转移只与上一阶段的状态有关，而且这样不用手动给j<vi的做转移
   3. for__(i,1,n)

```
    rof__(j,m,v[i])
        w[j]=max(w[j], //不选择第i个物品
            w[j-v[i]]+w[i]); //选择第i个物品
```

四. 变形
   1. 数字组合：给n个正整数ai，求选出任意个数，和为m的方案数
      i. 变量：阶段i为正在判断第几个数，状态j改为当前和，存储方案数
      ii. 初值：除了w[0]=1以外都是0

iii. 变形：因为不是求最大价值而是所有可能的方案数了，所以转移方程的max函数要改为加法函数

iv.
```
for__(i,1,n)
    rof__(j,m,a[i])
        w[j]+=w[j-a[i]]);
```

2. 手抓饼配料可能数：给n个配料及其大小v，大小不超过m的饼的方案数
    i. 变量：阶段i为正在判断第几个配料，状态j为当前饼的大小，存储大小为j时的方案数，p为消失的配料编号
    ii. 初值：每次除了d[0]=1以外都是0
    iii. 变形：转移方程为加法顺便对大数取余，每次输出d[0]到d[m]的和
    iv.
```
do{
    v[p]=m+1;       //让第p个配料不可能被选择
    memset(d,0,sizeof(d));
    d[0]=1;
    for__(i,1,n)
        rof__(j,m,v[i])
            d[j]=(d[j]+d[j-v[i]])%1000000007;
    rof__(j,m,1)
        d[0]=(d[0]+d[j]);
    printf("%llu\n",d[0]);
    scanf("%d",&p);
}while(k--);
```

3. 最少蛋糕数：n种蛋糕美味度w，想吃到美味度和为m但数量最少的蛋糕数
    i. 变量：阶段i为正在判断第i个蛋糕，状态j为该蛋糕使用次数，存储蛋糕数
    ii. 初值都除了d[0]=0外都大于n
    iii. 变形：转移方程为min
    iv.
```
for__(i,1,n)
    for__(j,1,g[i])
        if(j*w[i]>m)
            break;  //剪枝
        else
            rof__(k,m,w[i])  //01背包逆序
                d[k]=min(d[k],d[k-w[i]]+1);
```

4. 最小差值：k个数分成两组，求两个总和的最小差值
    i. 变量：v为当前数的大小，t[j]存储小于等于j的，最接近j的总和
    ii. 初值都是0
    iii.
```
for_(i,0,k){       //01背包，存储的权值为体积v
    scanf("%d",&v);
        rof__(j,n/2,v)
            t[j]=max(t[j],t[j-v]+v);}
cout<<n-2*t[n/2];//大-小=大+小-2*小=总-2*小
```

5. 光盘刻歌：在m张容量t的光盘存n首歌，各光盘内按曲顺排
    i. 除了考虑光盘内充足容量时的转移外，还要考虑从前一张光盘转移
    ii.
```
int n,t,m;       //曲数，光盘长度，光盘数
scanf("%d%d%d",&n,&t,&m);
for_(i,0,n)
    scanf("%d",l+i);
```

```
for_(i,0,n)        //遍历每首歌
    rof__(j,m,1){    //遍历每张碟
        rof__(k,t,l[i]){              //遍历每分钟
            d[j][k]=max(d[j][k]    //i歌不放（直接继承i-1歌）
                ,d[j][k-l[i]]+1);//i歌放在j光盘（继承j光盘）
            d[j][k]=max(d[j][k]    //为避免在j光盘重复计数i歌，后考虑从前一光盘转移
                ,d[j-1][t]+1);  //i歌放在j光盘（继承j-1光盘）
        }
}
cout<<d[m][t];
```

6. 录取概率：m所学校，学费a，录取率b，求最高录取率
   i. 变量：状态学校i，阶段学费j，存储用了学费j的最大录取率d
   ii. 转移方程是条件概率
   iii. for_(i,0,m)

```
    rof__(j,n,a[i])
        d[j]=max(d[j],d[j-a[i]]+(1-d[j-a[i]])*b[i]);
printf("%.1f%%",100*d[n]);
```

7. 摆花方法数：n种花各a[i]个，求按序摆各花摆满m盆的方法数
   i. 每种花都可以放[0,a[i]]次，需要多一层循环
   ii. 初值只有：d[0][0] = 1;
   iii. for__(i,1,n){

```
    cin>>a;
    for__(j,0,a)
        rof__(k,m,j)
            d[i][k] = (d[i][k] + d[i-1][k-j]) % p;}
cout<<d[n][m];
```

8. 多维背包：考虑选择物品数量同时也考虑体积
   i. 注意两维都是01背包，都要逆序
   ii. int n,m,s;

```
cin>>n>>m>>s; //数据量，容量1，容量2
int a,b;
for_(i,0,n){
    cin>>a>>b;       //体积2，价值
    rof__(j,m,1)
        rof__(k,s,a)
            if(d[j-1][k-a] + b > d[j][k])
                d[j][k] = d[j-1][k-a] + b,
                ans= max(ans, d[j][k]);}
```

9. 冰水挑战：n个挑战可以依次选择接不接受，x体力的你选择接受第编号i次挑战的话，x会变成min(x,bi)-ai+ci，选择不接受就直接变成x+ci，求最多能完成几次挑战
   i. 思路：状态肯定是挑战i，阶段如果选择当前体力，数组大小会不够用
   ii. 变量：以完成挑战次数作为阶段j，存储完成第j次挑战时的体力，则最后判断最大哪个j有体力即可找到答案j
   iii. 初值：其他为负数，d[0]=初始体力c0

iv. 转移：j非0且min(x,bi)>ai时可以从上一轮d[j-1]转（接受i挑战）

v. 转移2：上一轮的d[j]非0时可以从上一轮d[j]+=c[i]（不接受i挑战）

vi. 仅供参考的自己写的一维法：

vii.
```
memset(d, -1, sizeof(d));
d[0]=c0;
for__(i,1,n){
	for__(j,0,i){
		if(d[j]>0)		//i-1轮就可以有j次，选择放弃挑战i轮
			d[j]+= c[i];
		if(j&& min(d[j-1], b[i])> a[i])	//j-1>=0且不放弃i
			d[j]=max(d[j], min(d[j-1], b[i])- a[i]+ c[i]);}}
rof__(k,n,0)
	if(d[k]){
		cout<<k<<endl;
		break;}
```

viii. 一定能过的，从浩然的高中同学那找来的方法：

ix.
```
memset(d, 0, sizeof(d));
d[1][0]=C+c[1];
if(min(C, b[1])-a[1]>0)
	d[1][1]=min(C, b[1])- a[1]+ c[1];
for__(i,2,n){
	for__(j,0,i){
		if(j&& min(d[i-1][j-1], b[i])- a[i]>0)
			d[i][j]=max(d[i][j],
			min(d[i-1][j-1], b[i])- a[i]+c[i]);
		if(d[i-1][j]>0)
			d[i][j]=max(d[i][j], d[i-1][j]+c[i]);}}
rof__(k,n,0)
	if(d[n][k]){
		cout<<k<<endl;
		break;}
```

10. 读论文挑战：按顺序读n篇论文，各需要花ai时间去读，每读完一篇会增加1点灵感，可以通过扔掉2点灵感来使ai变成向上取整ai/2（只能变一次），可以通过扔掉3点灵感来不读第i篇论文，求t分钟最多能读多少篇论文

i. 分析：分钟数不太好存，可以将分钟数作为存储的内容，另外，维度灵感>2>3的判断蛮麻烦的，可以通过反向记录灵感+2，灵感+3解决

ii. 阶段i：读到第几篇论文，状态j：读完几篇论文，状态k：灵感数

iii. 注意：并不是i越大对应j就越大，干脆每次决策后都更新一下ans

iv.
```
memset(d, 0x3f, sizeof(d));
d[0][0][0]=0;
for__(i,1,n){
	scanf("%d",&a);
	for__(j,0,i)
		for__(k,0,j){
			//读i
			d[i][j+1][k+1]=min(d[i][j+1][k+1], d[i-1][j][k]+ a);
			//加快读i
			d[i][j+1][k+1]=min(d[i][j+1][k+1], d[i-1][j][k+2]+ (a+1)/2);
			//跳过i
			d[i][j][k]=min(d[i][j][k], d[i-1][j][k+3]);
```

```
            //更新答案
            if(d[i][j][k]<=t)
                ans=max(ans, j);}}
```

11. 掷几个骰子获得n的概率最大？骰子面上的数变成了d[6]={2,3,4,5,6,7};
    i. 骰子各面的权值作为物品容量，i个骰子投出k的方法数是sum（i-1个骰子投出k-权值），为了方便判断会不会越界或访问到上一轮不存在的数，对下标做加法并反向赋值
    ii. 注意：i轮要先赋0再从i-1轮求和转移，最方便的是开两维数组
    iii.

```
int ans=1;
double dn=1;    //denominator：分母
t[0][0]=1;      //起点!
for_(i,1,n){
    rof__(k,(i-1)*7,(i-1)*2)
        for_(j,0,6)
            t[i][k+d[j]]+=t[i-1][k];
    dn*=6;
    p[i]=t[i][n]/dn;
    if(p[i-1]>1e-10&& p[i]<=p[i-1])
        break;
    else
        ans=i;}
```